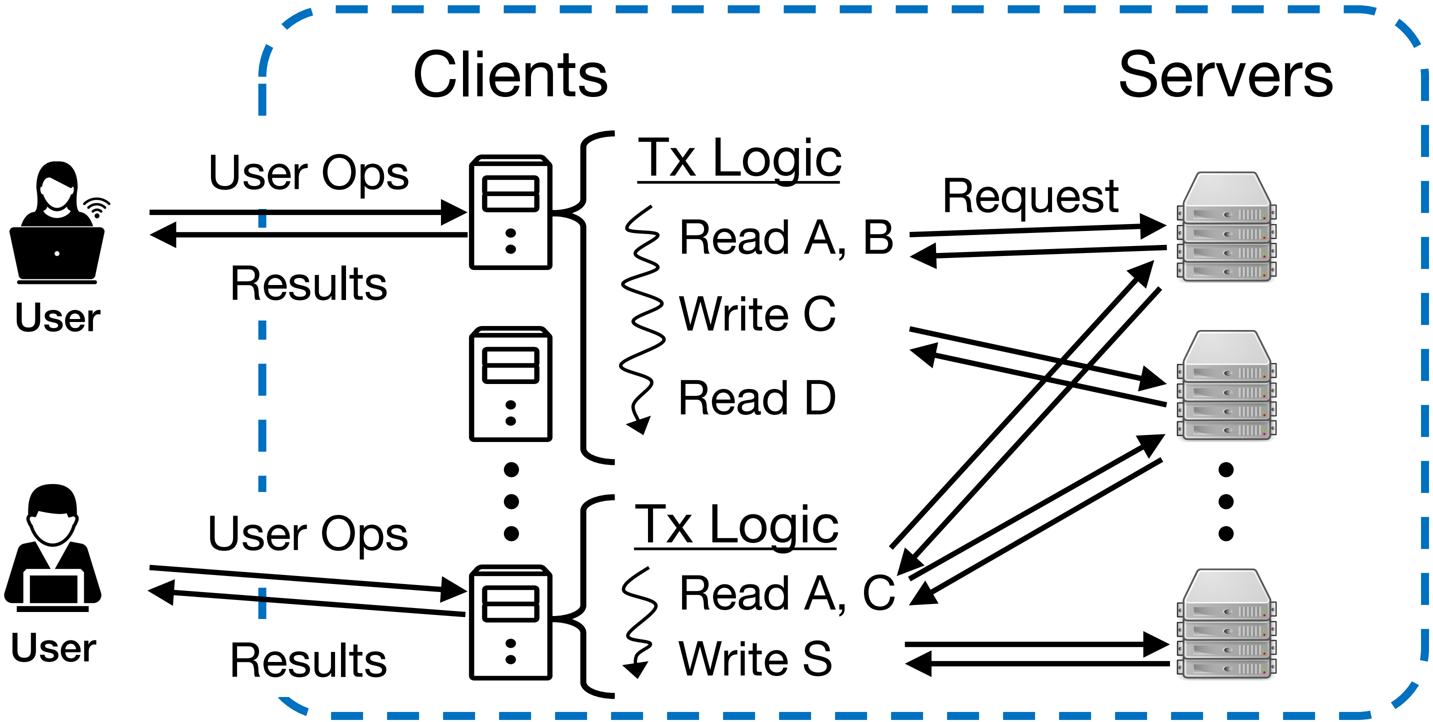
NCC: Natural Concurrency Control for Strictly Serializable Datastores by Avoiding the Timestamp-Inversion Pitfall

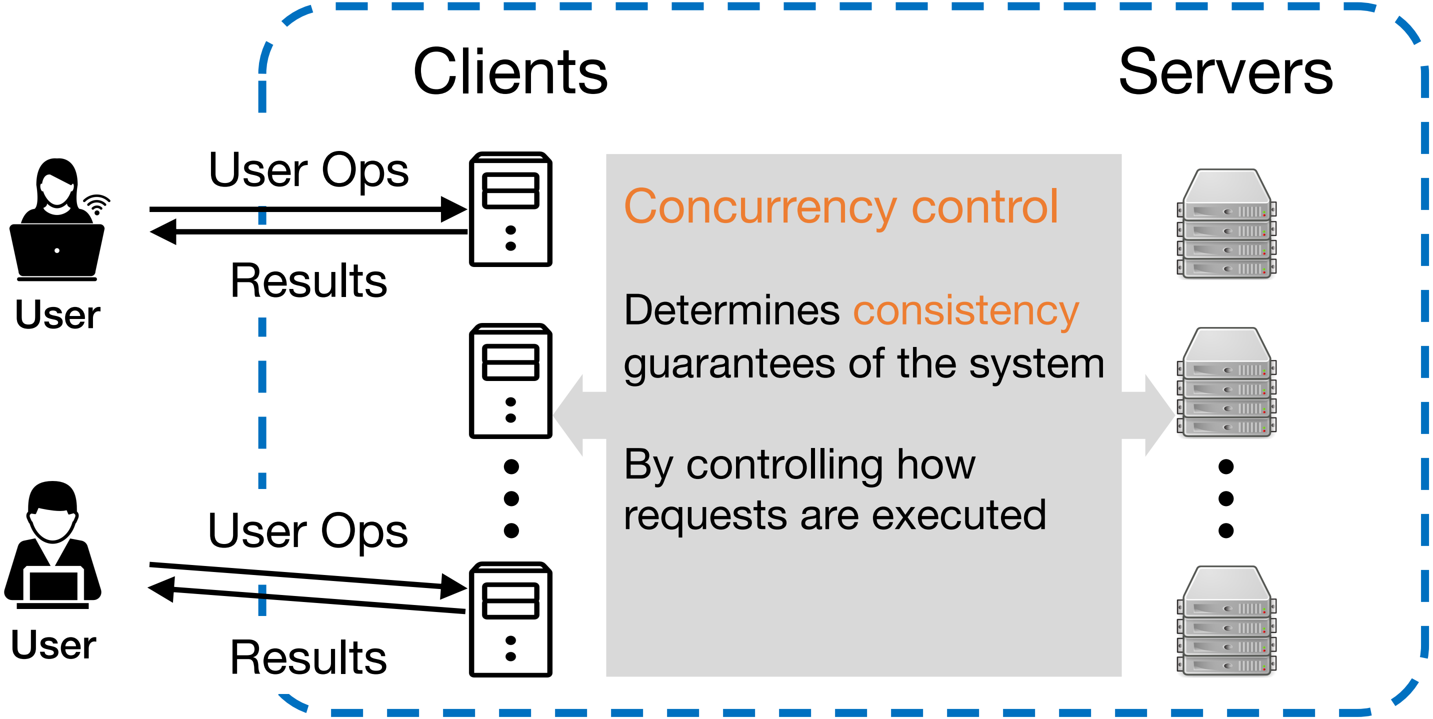
Haonan Lu,[★]
Shuai Mu,[♠] Siddhartha Sen,[♣] Wyatt Lloyd[◆]

[★]University at Buffalo [♠]Stony Brook University
[♣]Microsoft Research [◆]Princeton University

Transactional Datastores



Transactional Datastores



Strict Serializability

- Transactions take effect in a total order
  - Serializable: requests do not interleave
- Respects the real-time ordering
  - If tx_1 ends before tx_2 starts, then tx_1 must be ordered before tx_2 in the total order

Strict Serializability Is Costly

- Expensive mechanisms
  - Extra messages, locking, excessive aborts
  - Degrade system performance
- These costs are unnecessary for **naturally consistent** transactions

Natural Consistency

- Transaction requests arrive in an order that is already strictly serializable
- Prevalent in datacenter workloads
  - ➢ Many are reads: Interleaving is okay
  - ➢ Many are short: Interleaving is less likely
  - ➢ Many arrive in a real-time order: tx_1 ends before tx_2 starts, then tx_1's requests must arrive before tx_2's

Executing **naturally consistent** transactions simply in the order they arrive at servers naturally satisfies strict serializability

NCC: Natural Concurrency Control

- Guarantees strict serializability
- Leverages natural consistency
- Achieves minimal costs in common cases
  - ➢ One-round latency, lock-free, non-blocking execution

Three Pillars of Design

- Non-blocking execution
- Timestamp-based consistency checking
- Decoupled response management

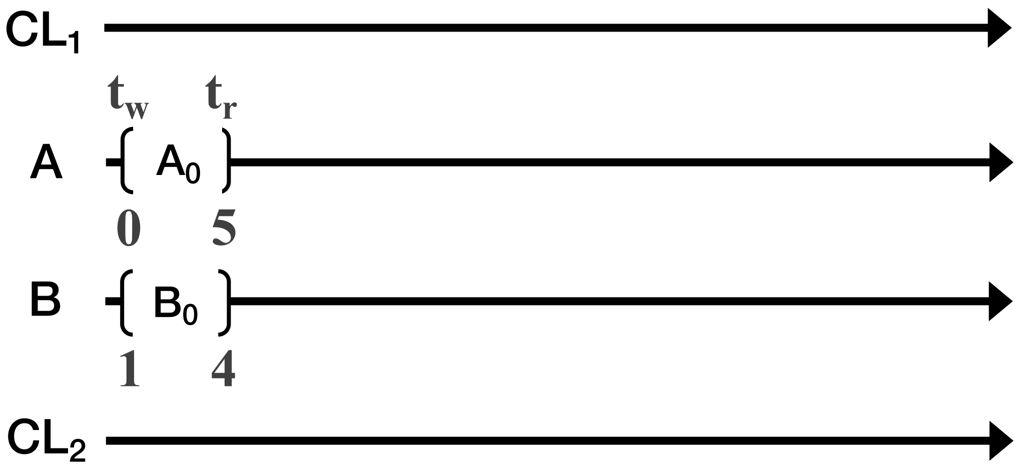
Non-blocking Execution

- Client pre-assigns timestamps, e.g., physical time
- Requests executed in the order they arrive
- Refine timestamps to match the arrival/execution order
- Immediately visible to subsequent transactions
- Responses are buffered, and sent when safe

Non-blocking Execution Example

tx_1 = {read A, write B}

tx_2 = {read B, write A}



Non-blocking Execution Example

tx_1 = {read A, write B}

tx_2 = {read B, write A}

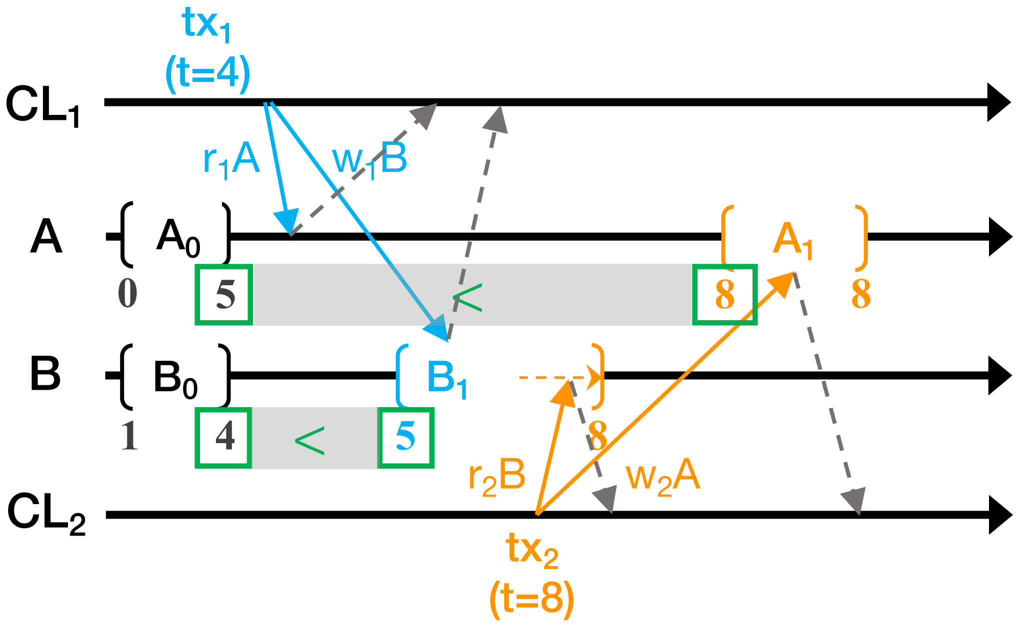
Buffered responses:

$tx_1.r_1A$ ←-- A_0, [0, 5]

$tx_1.w_1B$ ←-- "done", [5, 5]

$tx_2.r_2B$ ←-- B_1, [5, 8]

$tx_2.w_2A$ ←-- "done", [8, 8]



Safeguard

- Timestamp-based consistency checking
- Ensures a total order
  - [t_w, t_r] represents the time range where a request is valid
  - [t_w, t_r] pairs represent the arrival/execution order
  - The intersection of [t_w, t_r] pairs is a serialization point

Safeguard Example

Returned responses:

$tx_1.r_1A \leftarrow A_0$, [0, 5]

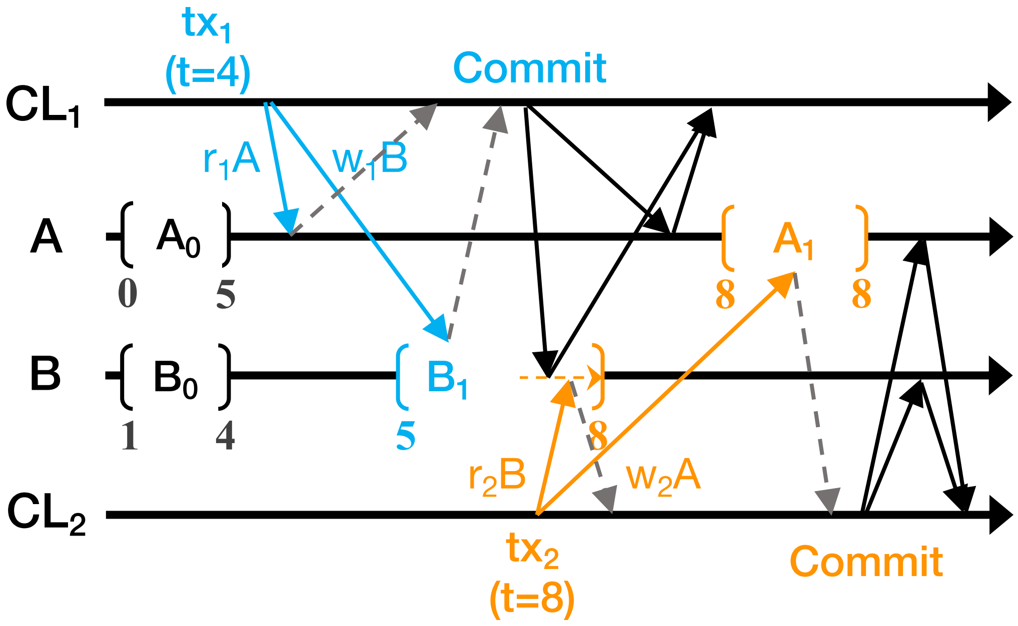
$tx_1.w_1B \leftarrow$ “done”, [5, 5]

$tx_1.t_{commit} = 5$

$tx_2.r_2B \leftarrow B_1$, [5, 8]

$tx_2.w_2A \leftarrow$ “done”, [8, 8]

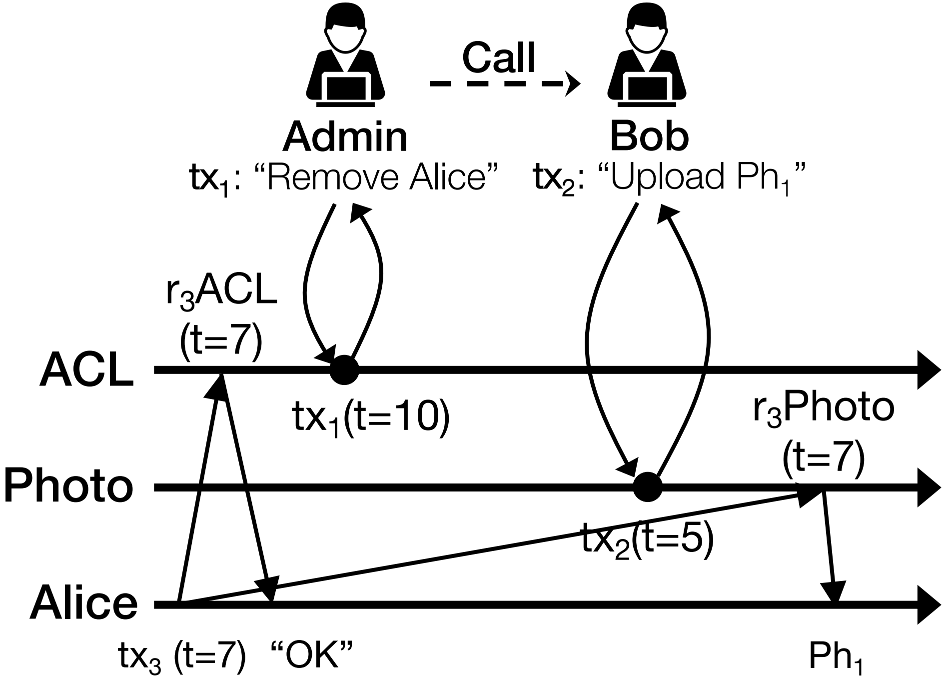
$tx_2.t_{commit} = 8$



Timestamp-Inversion Pitfall (TIP)

- Fundamental correctness pitfall in timestamp-based strictly serializable techniques
- Timestamps fail to guard against a total order that violates the real-time ordering between transactions in subtle cases

Example of Timestamp Inversion



Execution order is total:

$tx_2 \xrightarrow{exe} tx_3 \xrightarrow{exe} tx_1$

Incorrectly inverts $tx_1 \xrightarrow{rto} tx_2$

"Alice incorrectly sees Ph_1"

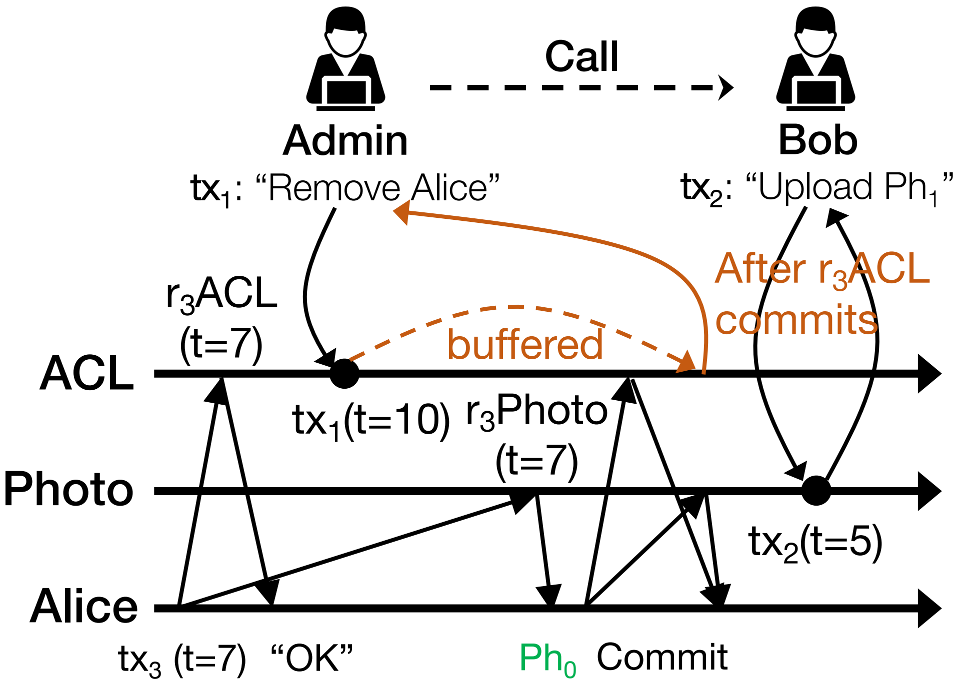
TIP is subtle: tx_3 interleaves with non-conflicting tx_1 & tx_2

TIP is fundamental: affects various types of transactions in multiple prior systems

Response Timing Control (RTC)

- Control when to send decoupled responses
- Disentangle the subtle interleaving in transactions’ real-time order
- No interference with non-blocking execution

RTC Avoiding TIP



tx_3 arrives before
tx_1 is responded before
Bob is notified before
tx_2 arrives

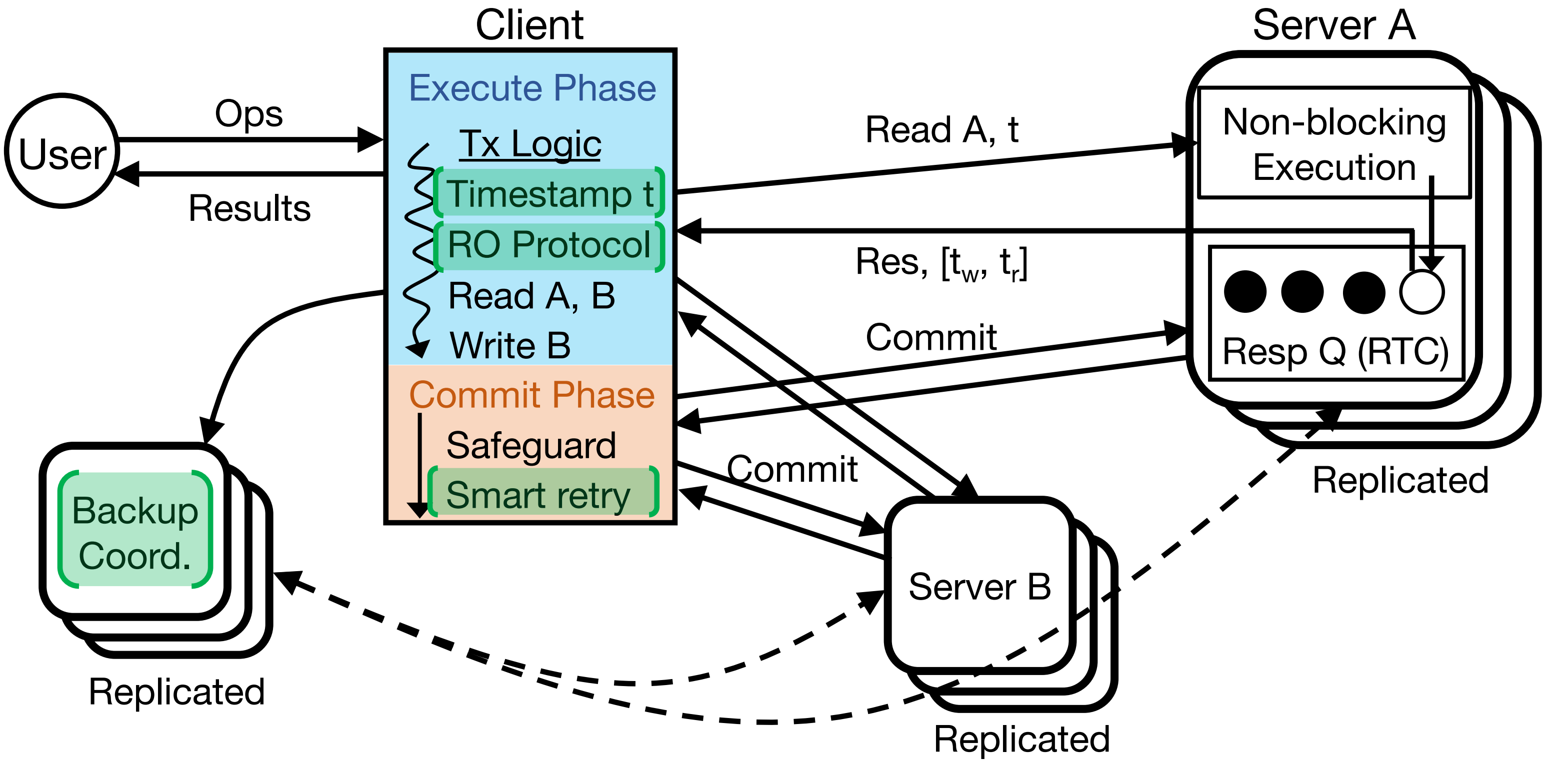
Execution order is total:

$tx_3 \xrightarrow{exe} tx_1 \xrightarrow{exe} tx_2$

Respects $tx_1 \xrightarrow{rto} tx_2$

"Alice sees Ph_0, not Ph_1"

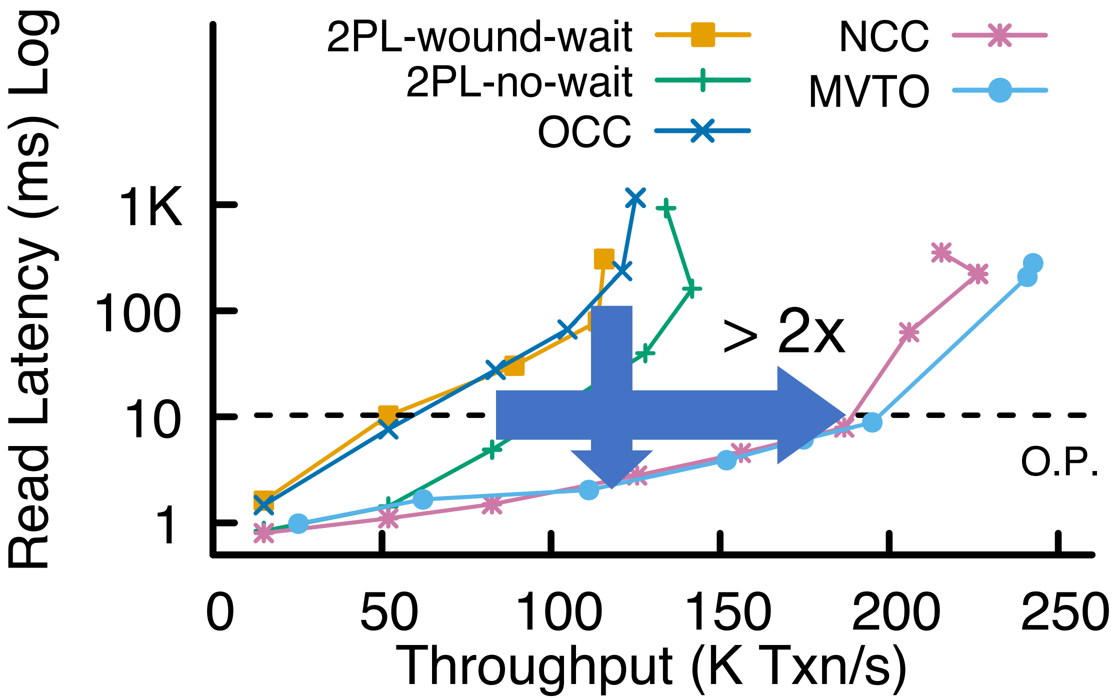
Architecture & Protocol Overview



Implementation and Evaluation

- Built on Janus's framework [OSDI '16]
- Baselines
  - Strictly serializable techniques, e.g., OCC and 2PL
  - Serializable protocols, e.g., MVTO (performance upper bound)
- Workloads
  - Synthetic Facebook-TAO and Google-F1 (read-dominated, one-shot)
  - TPC-C (many writes, multi-shot)
  - Varying write fraction in Google-F1 (write-intensive, one-shot)

Latency-Throughput, Google-F1



Conclusion

- NCC: Natural Concurrency Control
  - ➢ Minimal-cost, strictly serializable technique, leveraging natural consistency
- Timestamp-inversion pitfall
  - ➢ Correctness violation in timestamp-based strictly serializable techniques
- Implementation and evaluation of NCC for datacenter workloads
  - ➢ Significantly outperforms strictly serializable solutions
  - ➢ Closely matches the performance of serializable techniques

Thank you